DANIEL GOMES DE PINHO ZANCO

CLASSIFICAÇÃO DE CENAS ACÚSTICAS UTILIZANDO TÉCNICAS DE APRENDIZAGEM PROFUNDA

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Elétrica da Universidade Federal de Santa Catarina como requisito para aprovação da disciplina EEL7890 - Trabalho de conclusão de Curso (TCC).

Orientador: Eduardo Luiz Ortiz Batista.

Co-Orientador: Walter Antônio Gontijo.

FLORIANÓPOLIS
2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Zanco, Daniel Gomes de Pinho
Classificação de Cenas Acústicas Utilizando Técnicas de Aprendizagem Profunda / Daniel Gomes de Pinho Zanco ; orientador, Eduardo Luiz Ortiz Batista, coorientador, Walter Antônio Gontijo, 2018.
56 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2018.

Inclui referências.

1. Engenharia Elétrica. 2. Classificação de Cenas Acústicas. 3. Aprendizagem de Máquina. 4. Redes Neurais. 5. Aprendizagem Profunda. I. Ortiz Batista, Eduardo Luiz. II. Gontijo, Walter Antônio. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. IV. Título.

Daniel Gomes de Pinho Zanco

Classificação de Cenas Acústicas Utilizando Técnicas de Aprendizagem Profunda

Este Trabalho foi julgado adequado para obtenção do Título de Bacharel em Engenharia Elétrica e aprovado, em sua forma final, pela Banca Examinadora

Florianópolis, 27 de dezembro de 2018.

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Eduardo Luiz Ortiz Batista, Dr. Eng.
Orientador
Universidade Federal de Santa Catarina

Eng. Walter Antônio Gontijo, MSc.
Coorientador
Universidade Federal de Santa Catarina

Prof. Richard Demo Souza, Dr. Eng.
Universidade Federal de Santa Catarina

Eng. Felipe Dennis de Resende Oliveira, MSc.
Universidade Federal de Santa Catarina

Agradecimentos

Desejo expressar meu reconhecimento a todos que, de uma maneira ou outra, colaboraram na realização deste trabalho.

Aos meus pais, por todo o apoio emocional e incentivo ao longo dos anos.

Ao Laboratório de Circuitos e Processamento de Sinais (LINSE), pela oportunidade de realizar este trabalho.

Ao professor Eduardo Luiz Ortiz Batista, pela orientação na elaboração deste trabalho.

Ao meu coorientador Walter Antônio Gontijo, pelos anos de supervisão e aprendizado proporcionados dentro do LINSE.

Aos meus colegas do LINSE, pelas importantes discussões e trocas de ideias quanto ao trabalho.

RESUMO

Este trabalho apresenta um sistema de classificação de cenas acústicas. Em aplicações envolvendo o uso desse tipo de sistema, deve-se identificar, dentre um número limitado de categorias, o ambiente no qual gravações de áudio foram realizadas. São propostas alterações em um sistema de classificação *baseline*, que consiste de técnicas de aprendizagem profunda, a fim de obter um melhor desempenho. O desempenho do sistema proposto é avaliado sobre um conjunto de dados e comparado ao *baseline*, obtendo melhorias significativas.

Palavras-chave: Classificação de Cenas Acústicas. Aprendizagem de Máquina. Redes Neurais. Aprendizagem Profunda.

ABSTRACT

This work presents an acoustic scene classification system. In this application, the environment in which audio recordings were performed is identified from a limited number of categories. In order to achieve a better performance, modifications to a baseline classification system, which consists of deep learning techniques, are proposed. The proposed system's performance is evaluated on a dataset and compared to the baseline's, achieving significant improvements.

Keywords: Acoustic Scene Classification. Machine Learning. Neural Networks. Deep Learning.

Lista de Figuras

Lista de Tabelas

Sumário

CAPÍTULO 1

Introdução

Sons contêm uma grande quantidade de informação sobre ambientes e os eventos que neles ocorrem. Pessoas utilizam essas informações para entender o que acontece no seu entorno. Por exemplo, dentro de casa é possível perceber que há visitas ou notar se há trânsito movimentado em uma rua próxima, mesmo sem a informação visual da sala de estar ou olhando pela janela, respectivamente.

A habilidade de percepção auditiva pode ser considerada trivial no dia a dia das pessoas. No entanto, o mesmo não pode ser afirmado para computadores, uma vez que algoritmos capazes de reconhecer eventos sonoros ainda não são uma tecnologia plenamente desenvolvida. Esse tipo de tecnologia é de grande interesse em diversas aplicações. Por exemplo, sistemas de monitoramento poderiam reconhecer acidentes de trânsito, arrombamentos (a partir do som da quebra de um vidro) ou disparos de armas. Além disso, o monitoramento acústico pode ser vantajoso em ambientes com pontos cegos ou com iluminação variável, em que o monitoramento de vídeo teria dificuldades.

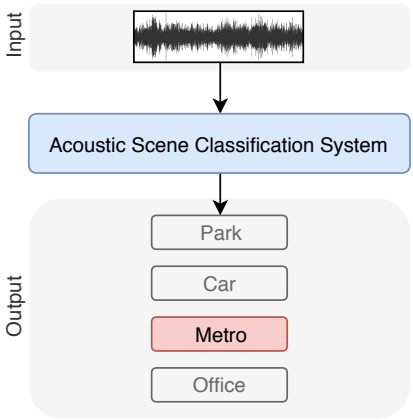
A análise automática de áudio também pode beneficiar sistemas de navegação robótica e dispositivos portáteis inteligentes. Esses dispositivos poderiam melhorar suas tomadas de decisões a partir de infor-

mações acústicas sobre o contexto do ambiente em que estão inseridos. Um *smartphone* poderia trocar de *ringtone* em ambientes mais formais, ou um carro poderia reduzir a velocidade ao "ouvir" crianças brincando nos arredores, por exemplo.

Os sons presentes no dia a dia das pessoas têm uma grande variedade acústica. Por isso, é comum que a definição manual de indicadores da presença dos sons que estão no cotidiano das pessoas seja uma tarefa difícil. Em alguns casos isso não é tão problemático, por exemplo, um disparo de arma pode ser identificado pela energia da sua pressão sonora. Entretanto, os padrões sonoros que definem certos ambientes, como os de um parque, não são facilmente determinados. Os métodos utilizados nesse tipo de problema geralmente se baseiam em técnicas de aprendizagem de máquina (*machine learning*, em inglês) [1, 2]. Em tais técnicas, os parâmetros do sistema são obtidos automaticamente a partir de conjuntos de exemplos dos sons alvo. Assim, torna-se possível que um computador "aprenda" a reconhecer padrões complexos em diversos tipos de sons.

Dentre os métodos de aprendizagem de máquina disponíveis, os baseados em aprendizagem profunda têm se destacado em diferentes áreas de inteligência artificial, obtendo avanços em aplicações como síntese e reconhecimento de fala [3, 4], reconhecimento de imagens [5] e processamento de linguagem natural [6].

Figura 1.1: Visão geral de um sistema de classificação de cenas acústicas.



Fonte: do Autor.

Neste trabalho, pretende-se utilizar técnicas de aprendizagem profunda para desenvolver um sistema de classificação de cenas acústicas. Nessa aplicação, deve-se identificar, dentre um número limitado de categorias, o ambiente no qual gravações de áudio foram realizadas. Um exemplo desse tipo de sistema é ilustrado na Figura 1.1, onde um trecho de áudio é processado e classificado como a cena acústica de um metrô.

1.1 Objetivos

Conforme mencionado anteriormente, o objetivo principal deste trabalho é desenvolver um sistema de classificação de cenas acústicas utilizando técnicas de aprendizagem profunda.

Como objetivos específicos, temos:

- Analisar algoritmos aplicados à classificação de cenas acústicas;
- Escolher um conjunto de dados com gravações de áudio rotuladas por ambiente;
- Extrair atributos do conjunto de dados;
- Treinar um classificador com algoritmos de aprendizagem profunda no contexto de classificação de cenas acústicas;
- Comparar o desempenho dos classificadores avaliados;
- Propor alterações no algoritmo do classificador a fim de obter um melhor desempenho.

1.2 Estrutura do Trabalho

O restante deste trabalho está organizado como descrito a seguir. No Capítulo 2, é realizada uma revisão sobre aprendizagem de máquina, aprendizagem profunda e classificação de cenas acústicas. O Capítulo 3 apresenta os detalhes do sistema de classificação de cenas acústicas desenvolvido. No Capítulo 4, são descritos os resultados obtidos a partir dos experimentos realizados. Por fim, o Capítulo 5 contém as conclusões obtidas e os trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica do presente trabalho. Assim, uma visão geral acerca dos conceitos de aprendizagem de máquina, aprendizagem profunda e classificação de cenas acústicas é apresentada.

2.1 Aprendizagem de Máquina

Muitas tarefas são complexas demais para serem resolvidas por algoritmos imutáveis desenvolvidos por seres humanos. As técnicas de aprendizagem de máquina são meios de obter a habilidade de realizar essas tarefas complexas [7].

A característica mais notável de algoritmos de aprendizagem de máquina é a sua capacidade de aprender a partir de dados [7]. Uma definição mais formal desse tipo de algoritmo é dada por Mitchell [8], que afirma "Diz-se que um programa de computador aprende com uma experiência E em relação a alguma classe de tarefas T e medida de desempenho P se o seu desempenho em tarefas T, como medido por P, melhora com a experiência E".

Dentre os tipos de tarefas mais comuns que utiliza-se algoritmos

de aprendizagem de máquina tem-se a regressão, na qual o algoritmo deve prever um valor numérico dada uma certa entrada, a síntese, que envolve a geração de amostras similares a um certo conjunto de dados, e a classificação, em que o algoritmo deve especificar a qual grupo pertence um certo exemplo [7].

2.1.1 A Tarefa de Classificação

Na tarefa de classificação, uma entrada é associada a uma dentre k categorias. Para isso, o algoritmo de aprendizagem deve especificar uma função $f : \mathbb{R}^n \longrightarrow \{1,...,k\}$. O classificador mapeia um vetor de entrada $\boldsymbol{x} \in \mathbb{R}^n$ em uma classe representada por um valor numérico $y \in \{1,...,k\}$, de forma que $y = f(\boldsymbol{x})$. No caso específico de $k = 2$, a classificação é chamada binária, para $k > 2$, diz-se que a classificação é multi-classe.

No caso da classificação multi-classe, é útil representar cada classe por um vetor binário $\mathbf{y} \in \{0, 1\}^k$. Nessa forma de representação, chamada de *onehot encoding*, um elemento do vetor é não-nulo e o resto é preenchido por zeros, o índice do elemento não-nulo indica a classe da respectiva amostra. Por exemplo, um vetor $(0, 1, 0)$ representa o rótulo da classe de número 2.

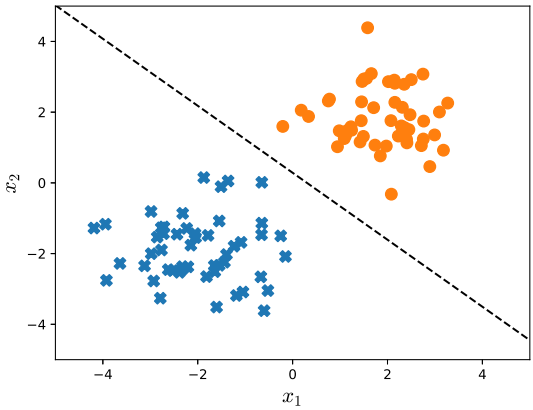
Na Figura 2.1 é mostrado um exemplo de classificação binária. Nele a entrada $\boldsymbol{x}$ possui duas dimensões (x_1 e x_2) e existem duas classes possíveis, uma indicada em azul e a outra em laranja. A linha pontilhada representa um limiar de decisão adequado para o classificador, a partir do qual os elementos das diferentes classes podem ser separados.

2.1.2 Conjunto de Dados

A experiência de um algoritmo de aprendizagem consiste em processar um conjunto de dados, ou *dataset*, relacionado a uma certa tarefa. O *dataset* é um conjunto de exemplos, que consistem em uma coleção de atributos medidos quantitativamente a partir de algum objeto ou evento que se deseja processar [7].

Conjuntos de dados são construídos de formas diferentes para diferentes tipos algoritmos de aprendizagem. Se os exemplos do *dataset* são associados a rótulos ou alvos, o algoritmo é dito supervisionado. Se o *dataset* somente contém os exemplos e seus atributos, o algoritmo é

Figura 2.1: Classificação binária.



Fonte: do Autor.

não-supervisionado.

2.1.3 Treinamento e Teste

No processo de aprendizagem, ou treinamento, é necessária a avaliação de uma métrica de desempenho específica à tarefa que o sistema está realizando. No caso de uma tarefa de regressão, o erro quadrático médio é muitas vezes utilizado como métrica de desempenho. Para a tarefa de classificação, geralmente é calculada a acurácia do modelo, ou seja, a proporção de exemplos que são corretamente classificados [7].

No contexto de aprendizagem de máquina, de maneira geral é mais interessante avaliar o desempenho do algoritmo considerado sobre um conjunto "desconhecido" de dados, ou seja, um *dataset* que não foi utilizado durante o treinamento, o qual é tipicamente chamado de conjunto de teste. Esse *dataset* permite determinar como seria a performance do sistema em um caso de uso real, isto é, na aplicação do algoritmo para avaliar novos casos. Portanto, é comum separar o *dataset* disponível para treinamento do algoritmo em um dois subconjuntos, sem intersecção, sendo eles de treinamento e de teste [7].

2.1.4 *Overfitting* e *Underfitting*

Considerando a divisão do *dataset* conforme descrito na seção anterior, verifica-se que um dos desafios principais em *machine learning* é obter um bom desempenho sobre os dados do conjunto de teste, os quais não foram utilizados durante o treinamento. Essa característica desejada é chamada de generalização [7].

Overfitting ocorre quando modelo de aprendizado se especializa demais no conjunto de treinamento e perde capacidade de generalização. Isso implica erro muito maior sobre o conjunto de teste que para o conjunto de treinamento. No caso de *underfitting*, o modelo não é capaz de aprender adequadamente com o conjunto de treinamento. Nesse caso o erro no conjunto de treinamento não é suficientemente baixo e diz-se que o modelo tem uma baixa capacidade de generalização [7].

2.1.5 Validação

Muitos algoritmos de *machine learning* buscam contornar os problemas de *overfitting* e *underfitting* utilizando parâmetros que não sofrem influência do processo de aprendizagem e são definidos antes do treinamento, os chamados hiperparâmetros. Esses parâmetros são usados para controlar a capacidade de aprendizagem e o comportamento dos algoritmos [7].

A escolha de valores dos hiperparâmetros é feita de forma a maximizar a capacidade de generalização do algoritmo. Entretanto, para obter uma estimativa adequada do erro de generalização, é importante que os exemplos de teste não sejam utilizados em decisões de treinamento [7]. Por causa disso, os exemplos de treinamento são subdivididos em um conjunto de validação e um novo conjunto de treinamento, especificamente voltado para encontrar os hiperparâmetros. Após o treinamento e a obtenção dos hiperparâmetros ótimos para o conjunto de validação, é estimado o erro de generalização a partir dos exemplos de teste.

2.1.6 Validação cruzada

No caso de um conjunto de dados com poucos exemplos, a validação cruzada é uma técnica eficiente para estimação do desempenho. Essa técnica envolve a criação artificial de múltiplas divisões de treinamento

e validação sobre um mesmo *dataset*, sendo o desempenho estimado a partir da média dos desempenhos sobre cada divisão.

Um dos métodos para realizar a validação cruzada é o *k-fold*. Tal método consiste em dividir o conjunto de dados aleatoriamente em k partições de mesmo tamanho, certificando-se de que não haja intersecção entre os conjuntos. Uma das partições é utilizada como conjunto de validação e as $k-1$ restantes são utilizadas no conjunto de treinamento. É realizado o treinamento de k modelos de aprendizado, um para cada partição, com conjuntos de validação sem intersecção entre si. Por fim, o desempenho de generalização é estimado pela média do desempenho dos k modelos sobre seus respectivos conjuntos de validação.

2.2 Aprendizagem Profunda

Os algoritmos tradicionais de aprendizagem de máquina funcionam bem para uma grande variedade de problemas mais simples. Entretanto, o seu desempenho não é satisfatório em tarefas mais complexas, tais como reconhecimento de fala ou visão computacional [7].

Além disso, os algoritmos tradicionais também sofrem com um aumento elevado de complexidade a medida que o número de dimensões do problema cresce, a chamada "maldição da dimensionalidade" [7]. Esse é um problema importante, visto a crescente quantidade de dados gerados e armazenados na era do *big data*.

O desenvolvimento da aprendizagem profunda (*deep learning*, em inglês) é motivado, principalmente, pela falha dos algoritmos de aprendizado tradicionais sobre os desafios citados [7].

Nos últimos anos, algoritmos baseados em aprendizagem profunda se tornaram o estado da arte na resolução de diversos problemas de inteligência artificial, tais como reconhecimento de fala [4] e processamento de linguagem natural [6]. Além disso, esses algoritmos têm sido capazes até mesmo de superar o desempenho humano em tarefas como a classificação de imagens [9].

Historicamente, aprendizagem profunda se originou da pesquisa de redes neurais artificiais [10]. Esses modelos de aprendizagem são inspirados na ideia de que, a partir da repodução de funcionalidades do cérebro humano, seria possível criar artificialmente um comportamento inteligente [7].

2.2.1 *Multilayer Perceptrons* (MLPs)

Multilayer Perceptrons (MLPs), também chamadas de redes neurais de alimentação direta (*feedforward*, em inglês), são os principais exemplos de modelos utilizados em aprendizagem profunda. O objetivo de uma MLP é aproximar uma função f^*, a qual é modelada por uma composição de funções, cujos parâmetros são aprendidos durante o treinamento da rede [7].

A composição de funções é feita com alimentação direta, ou seja, não há realimentação. Essas funções são conectadas em cascata, formando uma "rede", e cada função é chamada de camada. O uso de múltiplas camadas na arquitetura dessas redes é o que originou o termo "*deep learning*" [7].

Seja $\mathbf{y} = f^*(\mathbf{x})$ uma função que se deseja aproximar, uma rede neural com L camadas que a modele pode ser definida como

$$f(\boldsymbol{x}) = f_L(f_{L-1}(...f_2(f_1(\boldsymbol{x})))) = \hat{\boldsymbol{y}}, \tag{2.1}$$

onde $\hat{\boldsymbol{y}} \in \mathbb{R}^{n_L}$ é a predição da rede para uma entrada $\boldsymbol{x} \in \mathbb{R}^{n_0}$ e n_l é o número de unidades, ou largura, da camada l. As camadas da rede podem ser definidas por

$$f_l(\boldsymbol{a}^{[l-1]}) = \phi_l(\boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}) = \boldsymbol{a}^{[l]}, \tag{2.2}$$

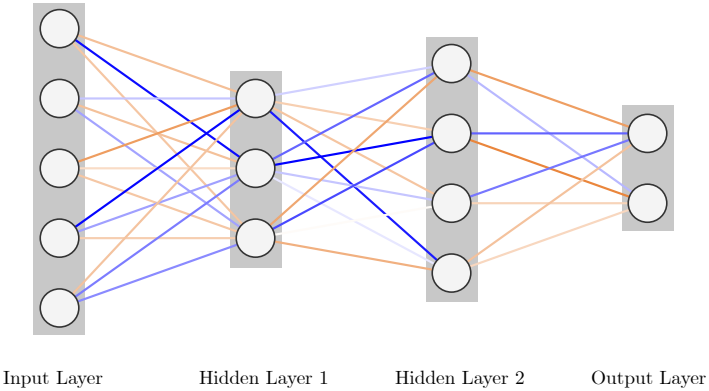
onde $\boldsymbol{W}^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ é a matriz de pesos, $\boldsymbol{b}^{[l]} \in \mathbb{R}^{n_l}$ é o vetor de viés (ou *bias*, em inglês), $\boldsymbol{a}^{[l]} \in \mathbb{R}^{n_l}$ é o vetor de ativações e $\phi_l(\cdot)$ é a função de ativação, todos referentes à camada l.

Na Figura 2.2 é mostrado um exemplo de representação da arquitetura de uma rede neural do tipo MLP. Cada círculo é uma unidade respectiva da camada e as conexões entre as unidades indicam a matriz de pesos. Por simplicidade, vetor de *bias* não é representado em tal figura.

2.2.2 Funções de Ativação

Funções de ativação são operadores não-lineares aplicados ponto-a-ponto sobre um vetor de ativações, na saída das camadas de uma rede neural. A escolha da função está relacionada ao tipo de dados e de tarefa.

Figura 2.2: Exemplo de uma rede neural *feedforward* com duas camadas ocultas.



Fonte: do Autor.

A função logística, ou sigmoide, é tradicionalmente utilizada para realizar predições sobre uma variável binária, retornando um valor que pode ser interpretado como uma probabilidade [7]. Tal função é definida para a i-ésima unidade do vetor de ativações da l-ésima camada como

$$\text{sigmoid}\left(\mathbf{a}^{[l]}\right)_i = \frac{1}{1+\exp\left(-a_i^{[l]}\right)}. \tag{2.3}$$

Para grandes valores de $|a_i|$, a função satura em ± 1, e só é sensível à entrada quando a_i está próximo de 0, o que pode causar dificuldades na sua otimização durante o treinamento. Por esse motivo, essa função é atualmente pouco usada em redes *feedforward*.

Uma função comumente utilizada como função de ativação é a *Rectified Linear Unit* (ReLU) [11]. Tal função é é amplamente escolhida por não apresentar problemas de saturação e por ser de simples implementação. Sua definição é dada por

$$\text{ReLU}\left(\mathbf{a}^{[l]}\right)_i = \max\left(0, a_i^{[l]}\right). \tag{2.4}$$

Em problemas de classificação multi-classe, a função *softmax* é geralmente utilizada como a ativação da última camada. Essa função pode ser vista como uma generalização da função logística para uma

distribuição de probabilidades sobre uma variável aleatória multidimensional. A função retorna um vetor de soma unitária e pode ser definida como

$$\text{softmax}\left(\mathbf{a}^{[l]}\right)_i = \frac{\exp\left(a_i^{[l]}\right)}{\sum_j \exp\left(a_j^{[l]}\right)}. \tag{2.5}$$

2.2.3 Redes Neurais Convolucionais (CNNs)

Redes neurais convolucionais (CNNs), ou *convnets*, são um tipo especial de rede neural que possui camadas baseadas na operação matemática de convolução, em contraste com a multiplicação matricial utilizada em MLPs. A operação de convolução é linear e tipicamente voltada para dados com uma estrutura em *grid*. Por exemplo, um sinal de áudio pode ser visto como um *grid* unidimensional de amostras temporais, enquanto uma imagem seria representada por uma *grid* bidimensional de *pixels*.

Em redes MLP, uma matriz de pesos relaciona cada unidade de entrada com todas a saídas. Já em redes convolucionais, as unidades de entrada tipicamente interagem com poucas unidades de saída, ou seja, são interações esparsas, o que requer um número menor de operações por camada. Além disso, a operação de convolução possui a característica de equivariância à translação. Ou seja, se entrada estiver deslocada de uma quantidade de posições, a saída também estará.

Outro ponto importante das CNNs é o compartilhamento de parâmetros. Nas MLPs, há um conjunto de pesos para cada posição da entrada, já no caso das CNNs os pesos são utilizados sobre todas as posições da entrada, aumentando a eficiência de armazenamento. Uma CNN pode ser vista como um caso especial de MLP no qual os pesos são compartilhados para múltiplas regiões do *grid* da entrada.

2.2.3.1 A Operação de Convolução

A operação de convolução discreta unidimensional entre dois sinais pode ser definida como

$$x[n] * w[n] = \sum_k x[k]w[n-k], \tag{2.6}$$

onde $x[n]$ é a entrada e $w[n]$ é o *kernel*, ou a resposta ao impulso de um filtro.

Frequentemente, a implementação da convolução em bibliotecas de *deep learning* é realizada pela operação de correlação cruzada, que é um pouco mais simples do ponto de vista de implementação [7]. A diferença em relação a convolução está no somatório, onde o *kernel* $w[k]$ tem seus índices incrementados no mesmo sentido que a entrada $x[k]$, de forma que a correlação cruzada pode ser definida como $x[n] * w[-n]$.

Na prática, a entrada da rede geralmente é um tensor. Por exemplo, uma imagem colorida possui 3 canais de cores (RGB) para cada pixel e pode ser vista como um tensor tridimensional, com duas dimensões espaciais e uma sobre os canais de cor. Nesse caso, a operação realizada por uma rede convolucional bidimensional mapeia um tensor tridimensional na entrada em outro tensor tridimensional na saída.

Além disso, é importante que a operação de convolução realize implicitamente um processo de *zero-padding* para impedir que a unidade de saída encolha. São três os modos implícitos de *zero-padding* geralmente implementados, seguindo a terminologia do *software* MATLAB [12]: *valid*, *same* e *full*. Na convolução do tipo *valid*, não é feito *padding*, sendo a saída reduzida pelo tamanho do *kernel* menos um. No caso *same*, é feito *padding* suficiente para que o saída tenha o mesmo tamanho da entrada. Por fim, a convolução completa, ou *full*, realiza o *padding* de forma deixar o *kernel* processar cada elemento o mesmo número de vezes, aumentando o tamanho da saída pelo tamanho do *kernel* menos um.

2.2.3.2 *Pooling*

Após a operação de convolução e a aplicação da não-linearidade pela função de ativação, a saída de redes convolucionais geralmente é ainda modificada por uma camada, ou função, de *pooling*. Essa operação retorna certas sínteses estatísticas de regiões no entorno de cada unidade de saída. Podem ser calculados o valor máximo (*max pooling*) ou a média (*average pooling*) dentre uma região retangular da saída, por exemplo.

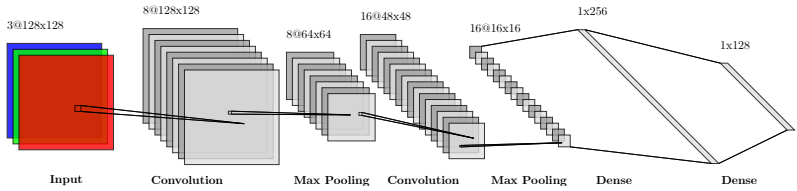
Camadas de *pooling* ajudam a tornar o modelo invariante à pequenas translações da entrada. Ou seja, se a entrada da rede neural sofrer uma pequena translação, a saída da camada de *pooling* deve ser aproxi-

madamente a mesma. A invariância à translação é uma característica útil em cenários em que a simples determinação da presença de certo padrão é mais importante que a sua localização exata, como é o caso de muitas tarefas de classificação.

A operação de *pooling* "resume" a informação das ativações sobre uma região, o que permite que menos unidades de saída sejam utilizadas. Em muitas implementações, o *pooling* usado para retornar sínteses estatísticas de regiões espaçadas de n posições em vez de 1. Isso beneficia a eficiência computacional de modelos profundos, reduzindo em n vezes o número de unidades para processar na camada seguinte. Se a próxima camada for do tipo MLP, essa redução ainda resulta em uma melhora na eficiência estatística e de armazenamento de parâmetros, dado que a quantidade de parâmetros de uma camada densa depende do tamanho da sua entrada.

A Figura 2.3 mostra um exemplo de arquitetura de redes neural profunda utilizada em problemas de classificação. A entrada da rede é uma imagem RGB e ela é composta por camadas convolucionais, *pooling* e densas.

Figura 2.3: Exemplo de uma rede neural profunda com camadas convolucionais.



Fonte: do Autor.

2.2.4 Treinamento

Como visto nas Subseções 2.1.2 e 2.1.3, a etapa de treinamento de uma rede neural convencional envolve processar um *dataset* como forma de "aprendizagem através do exemplo". Isso não é diferente para redes profundas, tendo o processo de aprendizagem o objetivo de otimizar uma medida de desempenho P em relação ao conjunto de teste, o que não pode ser feito diretamente.

Em vez disso, a otimização dos parâmetros $\boldsymbol{\theta}$ do modelo é feita a partir de uma função custo $J(\boldsymbol{\theta})$, com a expectativa de que o desempenho P melhore. A função custo para um conjunto de N exemplos de treinamento é definida como

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right), \tag{2.7}$$

onde $\hat{\boldsymbol{y}}$ é a predição do modelo, $\boldsymbol{y}$ é o rótulo de um exemplo e $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ é uma função perda específica da tarefa.

Para tarefas de classificação com K classes, a função perda geralmente utilizada é a entropia cruzada, definida como

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) + (1 - y_k) \log(1 - \hat{y}_k), \tag{2.8}$$

onde $\hat{y}_k$ e y_k são elementos dos vetores $\hat{\boldsymbol{y}}$ e $\boldsymbol{y}$, respectivamente.

2.2.4.1 *Back Propagation*

O processo de cálculo de uma estimativa $\hat{\boldsymbol{y}}$ a partir de uma entrada $\boldsymbol{x}$ em uma rede neural *feedforward* é chamado de propagação direta. A informação $\boldsymbol{x}$ se propaga da entrada, por unidades ocultas, até a saída $\hat{\boldsymbol{y}}$.

Durante o treinamento, é necessário calcular o gradiente de $J(\boldsymbol{\theta})$ para posterior uso em algoritmos de otimização. Propagação reversa (ou *back propagation*, em inglês) é um algoritmo que flui a informação da função custo de volta pelas camadas da rede, permitindo a computação desse gradiente. O algoritmo de *back propagation* basicamente utiliza a regra da cadeia para calcular as derivadas de forma eficiente.

2.2.4.2 Algoritmos de Otimização

Conjuntos de treinamento com grandes quantidades de dados são necessários para obter bons desempenhos de generalização. Entretanto, tais conjuntos resultam em custos computacionais elevados no processo de treinamento. Tal problema é levado em consideração pelos algoritmos de otimização utilizados em aprendizagem profunda.

Em geral, a otimização em aprendizagem profunda utiliza algoritmos baseados no método do gradiente. Em tal método, deve-se encontrar o mínimo da função custo $J(\boldsymbol{\theta})$ caminhando, com uma taxa de aprendizagem ϵ, na direção negativa do gradiente $\nabla J(\boldsymbol{\theta})$. Em geral, não é viável calcular o gradiente a partir de todo o conjunto de treinamento, por isso geralmente é utilizada uma variação desse algoritmo.

O *stochastic gradient descent* (SGD), ou método do gradiente estocástico, realiza uma estimativa do gradiente para um subconjunto de exemplos, ou *minibatch*, visando obter um treinamento eficiente para um grande conjunto de dados. O *minibatch* é amostrado do *dataset* de forma aleatória e sem reposição até que todo o conjunto seja processado. Uma passagem pelo conjunto de treinamento é chamada de "época". São necessárias muitas épocas para treinar uma rede neural, mas o número exato se altera com a capacidade da rede, o número de exemplos e a taxa de aprendizagem, entre outros fatores.

A taxa de aprendizagem é um hiperparâmetro que influencia o comportamento do treinamento e, por consequência, afeta o desempenho do modelo. A escolha da taxa de aprendizagem pode ser difícil, valores muito baixos tendem a tornar o treinamento lento, enquanto que valores muito altos podem trazer problemas de estabilidade na convergência do algoritmo [7]. Existem algumas variações do SGD que determinam a taxa de aprendizagem de forma adaptativa, a cada *minibatch*. Dentre esses algoritmos, pode-se citar o Adam [13].

2.2.4.3 Normalização de Atributos

Um procedimento importante para que a convergência dos algoritmos de otimização não seja lenta é normalização dos atributos na entrada de uma rede neural [14]. A normalização é comumente realizada pré-processando cada atributo, de forma que o valor médio e o desvio padrão, sobre todo o conjunto de treinamento, sejam aproximadamente nulo e unitário, respectivamente [15]. Desse modo, a normalização de um conjunto de N exemplos de treinamento com atributos k-dimensionais $\boldsymbol{x}^{(i)}$ pode ser calculada por

$$x_k'^{(i)} = \frac{x_k^{(i)} - \mu_k}{\sigma_k}, \tag{2.9}$$

onde a média μ_k é definida como

$$\mu_k = \frac{1}{N} \sum_{i=1}^{N} x_k^{(i)}, \tag{2.10}$$

e o desvio padrão σ_k é dado por

$$\sigma_k = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(x_k^{(i)} - \mu_k \right)^2}. \tag{2.11}$$

2.2.4.4 Inicialização

O treinamento de modelos de aprendizagem profunda em geral é feito por algoritmos iterativos e, portanto, exige que se especifique um ponto inicial de onde a otimização começa. A escolha do método de inicialização afeta consideravelmente a convergência no treinamento, a qual é normalmente difícil.

Uma das poucas propriedades conhecidas sobre como o ponto inicial influencia no treinamento é que deve haver uma "quebra de simetria" dos parâmetros iniciais. Se múltiplas unidades de uma camada tiverem os mesmos valores iniciais, o algoritmo de otimização irá atualizar essas unidades aproximadamente da mesma maneira. Isso tornaria as unidades redundantes e reduziria a capacidade do modelo.

Por esse motivo, a inicialização de camadas *feedforward* é feita com pesos aleatórios. Em geral, os valores do vetor de *bias* são definidos como constantes e somente a matriz de pesos é inicializada de forma aleatória. Uma heurística de inicialização de pesos de uma camada MLP com m entradas e n saídas é sugerida por Glorot e Bengio [16]

$$W[i,j] \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}} \right), \tag{2.12}$$

onde $U(a,b)$ é a distribuição uniforme com intervalo $[a,b]$.

2.2.4.5 *Data Augmentation*

Frequentemente, as grandes quantidades de dados requeridas para o treinamento eficiente de modelos profundos não estão disponíveis. Uma abordagem para contornar esse problema é gerar dados artificiais e

adicioná-los ao conjunto de treinamento. Esse processo, denominado *data augmentation*, tem sido utilizado com sucesso em diversas tarefas de aprendizagem de máquina, tais como reconhecimento de voz [17] e detecção de objetos [18].

No contexto de uma tarefa de classificação, uma das características desejáveis é a invariância a certas deformações dos atributos $\mathbf{x}$ de uma certa classe $\mathbf{y}$. Assim, um processo de *data augmentation* pode ser feito de forma simples a partir de pares $(\mathbf{x}, \mathbf{y})$ do *dataset* original, bastando realizar transformações nas entradas $\mathbf{x}$.

2.2.4.6 *Mixup*

Mixup é uma forma de *data augmentation* que independe do domínio ao qual os dados pertencem. Exemplos virtuais são gerados a partir da interpolação linear entre dois exemplos e seus rótulos, amostrados aleatoriamente do conjunto de treinamento [19]. Novos pares atributo/rótulo $(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}})$ são gerados pela equação dada por

$$\begin{aligned} \widetilde{\mathbf{x}} &= \lambda \mathbf{x}^{(i)} + (1-\lambda)\mathbf{x}^{(j)}, \\ \widetilde{\mathbf{y}} &= \lambda \mathbf{y}^{(i)} + (1-\lambda)\mathbf{y}^{(j)}, \end{aligned} \tag{2.13}$$

onde $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ e $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})$ são dois exemplos escolhidos aleatoriamente do conjunto de treinamento, e $\lambda \in [0, 1]$ é uma variável aleatória amostrada da distribuição Beta(α, α), para $\alpha \in (0, \infty)$.

Entre as vantagens do *mixup* estão a facilidade de implementação e a eficiência computacional. O *mixup* demonstradamente melhora a generalização de arquiteturas de redes neurais em problemas de classificação de imagem de *datasets* como *CIFAR* e *Imagenet* [19].

2.2.4.7 *Batch Normalization*

Durante o treinamento de uma rede neural, geralmente os exemplos na entrada são normalizados para que a otimização não seja lenta, por ocorrer de modo desigual através das dimensões dos atributos. *Batch normalization* estende a ideia de normalizar atributos às ativações de camadas ocultas de uma rede neural. Durante o treinamento, esse procedimento realiza a estimação da média e desvio padrão para cada *mini-batch* e aplica uma transformação que mantém a média das ati-

vações próxima de 0 e o desvio padrão próximo de 1. É mostrado experimentalmente que o uso de *batch normalization* pode acelerar o treinamento de redes neurais profundas [20].

2.2.4.8 *Dropout*

Dropout é um procedimento realizado para reduzir o *overfitting* de camadas de redes neurais. Esse procedimento tem o objetivo de impedir que as unidades de uma camada criem uma interdependência de seus pesos, o que reduziria a capacidade de generalização da rede.

O procedimento de *dropout* é diferente durante as etapas de treinamento e de teste. Na etapa de treinamento, para cada iteração do processo de otimização, unidades da rede neural são descartadas, cada uma com probabilidade p de ter seu valor temporariamente definido como 0. Na etapa de teste, todas unidades são utilizadas, mas são escaladas por p para compensar os pesos nulos durante o treinamento. De modo geral, pode-se interpretar o *dropout* como uma forma de regularização que adiciona ruído às unidades ocultas de uma rede neural [21].

2.2.4.9 *Early Stopping*

Redes neurais sofrem de *overfitting* no decorrer de muitas épocas de treinamento. *Early stopping* é uma forma simples de se evitar *overfitting* que pode, muitas vezes, ser superior a certos métodos de regularização [22]. Uma técnica básica de *early stopping* pode ser descrita pelos seguintes passos:

(*i*) Separar os dados de treinamento em conjuntos de treinamento e validação.

(*ii*) Treinar no conjunto de treinamento e, a cada época, avaliar o erro do modelo no conjunto de validação.

(*iii*) Parar de treinar se o erro de validação não melhorar, com uma "paciência" de uma quantidade pré-definida de épocas.

(*iv*) Utilizar os pesos da rede neural com o melhor desempenho no conjunto de validação.

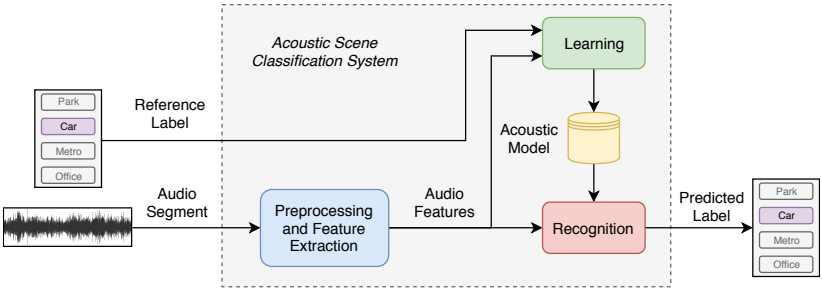
2.3 Classificação de Cenas Acústicas

Cenas acústicas se referem ao som que é formado quando sons de diferentes fontes, tipicamente de um cenário real, se combinam para formar uma mistura [1]. Por exemplo, a cena acústica de um parque pode ser formada por um conjunto de sons de pássaros, árvores ao vento e crianças brincando.

Sistemas de classificação de cenas acústicas realizam a tarefa de categorizar uma gravação de áudio dentre um conjunto de cenas pré-definidas. A maioria desses sistemas têm abordagens baseadas em aprendizagem supervisionada. Como discutido na Seção 2.1.2, essa abordagem requer um conjunto de dados pré-rotulados. A partir do *dataset* é realizado o pré-processamento e a extração de atributos dos dados, para posterior treinamento do modelo de aprendizagem.

Na Figura 2.4, é descrita uma visão geral de um sistema de classificação de cenas acústicas.

Figura 2.4: Visão geral de um sistema de classificação de cenas acústicas.



Fonte: do Autor.

2.3.1 Extração de Atributos de Sinais de Áudio

Sons são ondas mecânicas que se propagam através de um meio de transmissão. Para que possam ser analisados, é feita a conversão do som em um sinal elétrico $x(t)$ através de um microfone. Esse sinal analógico $x(t)$ é filtrado e convertido em um sinal discreto e digital $x[n]$, com uma determinada taxa de amostragem e nível de quantização.

A forma de onda de sinais áudio é de difícil análise. Geralmente, é quase impossível diferenciar cenas acústicas somente a partir da sua

representação no domínio do tempo [1]. Assim sendo, representações no domínio da frequência e tempo-frequência, como um modo de extração de atributos, são importantes para que o algoritmo de classificação alcance o desempenho desejado.

Além disso, sinais de áudio podem facilmente se tornar muito grandes, por exemplo, uma gravação estéreo de 10 segundos com uma taxa de amostragem de 48 kHz é composta por quase 1 milhão de amostras. Algoritmos de aprendizagem se tornam muito complexos ao tratar exemplos com tal elevado número de amostras. Assim, é importante que seja realizada a redução de dimensionalidade dos sinais, o que pode ser obtido a partir de extração de parâmetros.

2.3.1.1 Transformada de Fourier e Espectrograma

Em geral, é útil conhecer as características em frequência de uma cena acústica antes de realizar qualquer análise. A transformada de Fourier permite que se obtenha a representação em frequência, ou espectro, de um sinal.

Seja $x[n]$ um sinal discreto de tamanho N, a transformada discreta de Fourier (DFT) de $x[n]$ pode ser definida como

$$X[k] \triangleq \sum_{n=0}^{N-1} x[k] e^{-j(2\pi n/N)k}. \tag{2.14}$$

Ao aumentar o número de amostras N, é possível obter um espectro mais detalhado, com uma resolução de frequências melhor. Entretanto, é importante garantir que as amplitudes e frequências dos componentes senoidais do sinal $x[n]$ não variem na janela de N amostras. Essas são garantias necessárias ao utilizar a DFT para análises.

Na prática, o conteúdo de frequência da maioria dos sinais de áudio apresentam variações temporais. Logo, o número de amostras considerado para a DFT deve ser suficientemente pequeno para assegurar que o espectro não varie significativamente no decorrer das amostras.

Para contornar esse problema, utiliza-se uma representação no domínio do tempo-frequência, em que o sinal é dividido em trechos, ou *frames*, menores e é calculada a DFT de cada um deles. Pode-se definir

a transformada discreta de Fourier de tempo curto (STFT) [23] como

$$X[n,k] = \sum_{m=0}^{L-1} w[m]x[nH+m]e^{-j(2\pi k/N)m}, \tag{2.15}$$

onde $w[m]$ é uma função de janelamento de tamanho L e H é o tamanho do salto entre o início de cada *frame*. A função janela $w[n]$ é utilizada para reduzir alguns efeitos de vazamento espectral e para garantir a continuidade nas bordas dos *frames*. Além disso, o tamanho de salto H escolhido é, em geral, menor que o tamanho do *frame* L. Isso introduz uma sobreposição (ou *overlap*, em inglês) entre *frames* adjacentes, deixando as transições temporais mais suaves.

A magnitude do espectro $X[n,k]$, também chamada de espectrograma, permite que as variações no espectro de um sinal sejam analisadas de forma visual. Por sua estrutura multidimensional, é possível considerar o espectrograma como sendo uma imagem 3D: um canal de cor, uma dimensão temporal e outra de frequências. Na Figura 2.5 é mostrada a forma de onda e o espectrograma de uma cena acústica. Nessa escala de cores, as mais claras representam componentes de frequência com maiores amplitudes.

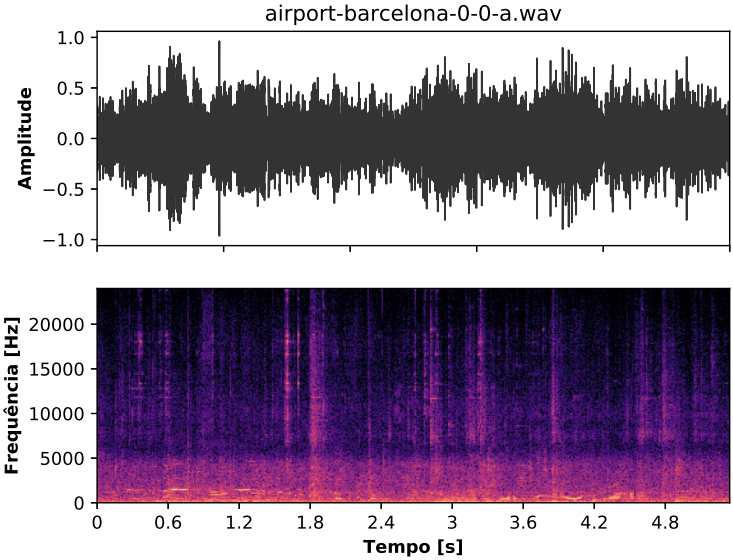
2.3.1.2 Escala *mel*

A escala *mel* é uma escala perceptual não-linear de frequências julgadas serem equidistantes por ouvintes [24]. Como a sua construção é experimental, existem várias expressões que a descrevem. Uma equação que descreve a escala *mel* em função de uma frequência f, em Hz, é dada por [25]

$$\text{mel}(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right). \tag{2.16}$$

Um atributo comumente utilizado em sistemas de classificação de cenas acústicas é o *log mel spectrogram*, também referido como *log mel-band energies*. Esse tipo de atributo pode ser extraído através da aplicação de um banco de filtros sobre a magnitude de um espectrograma e o posterior cálculo do logaritmo desses valores. Na Figura 2.6 é mostrado um exemplo de banco de filtros que realiza o cálculo da energia de 13 bandas mel.

Figura 2.5: Forma de onda e respectivo espectrograma de uma cena acústica de um aeroporto.



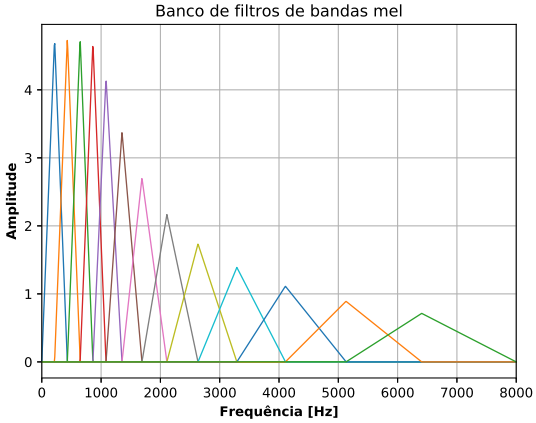
Fonte: do Autor.

2.3.2 Avanços Recentes

Tradicionalmente, a análise de cenas acústicas tem se baseado em técnicas de aprendizagem de máquina em conjunto com a extração de atributos de alto nível de abstração, com a chamada "engenharia de atributos" [1, 2]. Esse procedimento depende muito de um conhecimento especializado sobre as cenas em questão, podendo não generalizar bem. Mais recentemente, com o desenvolvimento de sistemas de análise de áudio baseados em aprendizagem profunda, atributos mais simples têm sido preferíveis [26].

Nos últimos anos, também houve a publicação de diversos conjuntos de dados de sons ambientais, tais como *AudioSet* [27], ESC-50 [28] e *UrbanSound* [29]. Com a maior facilidade de acesso a conjuntos de dados, torna-se possível a organização de competições para avaliar o desempenho de diversos sistemas de forma padronizada, além de facilitar o surgimento de novas ideias. É o caso de uma competição organizada

Figura 2.6: Resposta em frequência de um banco de filtros de bandas mel.



Fonte: do Autor.

por [30], na qual foi observado que a combinação de diferentes classificadores foi utilizada em muitos dos melhores sistemas. Também é sugerido que combinar classificadores treinados com esquemas de validação cruzada produz melhorias significativas

2.3.2.1 *Detection and Classification of Acoustic Scenes and Events* (DCASE)

Um dos eventos científicos de grande importância para a pesquisa em classificação de cenas acústicas é o *Detection and Classification of Acoustic Scenes and Events* (DCASE) [31, 32, 33], que é realizado desde o ano de 2013. Esse evento é composto de um *workshop*, onde são apresentados trabalhos por meio de palestras e pôsteres, e um desafio (ou *challenge*, em inglês), que envolve a realização de tarefas de aprendizagem de máquina aplicadas à análise de cenas e eventos acústicos. O desafio DCASE permite que pesquisadores compitam no desenvolvimento de sistemas de detecção e classificação a partir de conjuntos de dados padronizados.

Além disso, é relevante notar que no evento de 2017, na tarefa de classificação de cenas acústicas, os dois melhores sistemas utilizaram classificadores baseados em redes neurais convolucionais e *log mel energies* como atributos [34, 35].

CAPÍTULO 3

Desenvolvimento

Neste capítulo são apresentados o conjunto de dados utilizado e os detalhes do sistema de classificação de cenas acústicas desenvolvido.

3.1 Conjunto de Dados Utilizado

O *dataset* escolhido para o desenvolvimento do sistema de classificação de cenas acústicas foi o *TUT Urban Acoustic Scenes 2018* (Development dataset), referido como TUT2018 [36]. Esse conjunto de dados é disponibilizado pelos organizadores do DCASE 2018 *Challenge* [37], sendo ele construído a partir de gravações realizadas em diferentes locais de seis cidades europeias. O áudio está dividido em trechos de 10 segundos, com indicações de cidade, cena acústica e local de gravação. Os trechos são estéreo, com taxa de amostragem de 48 kHz e 24 bits de resolução, e há 864 segmentos de cada cena acústica (144 minutos de áudio), totalizando 24 horas de gravações.

Além disso, o *dataset* é subdividido em conjuntos de treinamento e teste, de forma que o conjunto de treinamento contém aproximadamente 70% dos exemplos. Dos 8640 segmentos, 6122 estão inclusos no conjunto de treinamento e 2518 no de teste. Na Tabela 3.1 são descritas

as 10 classes de cenas acústicas que fazem parte do *dataset*.

Tabela 3.1: Cenas acústicas incluídas no *dataset* utilizado e seus rótulos.

Cena acústica	**Label**
Aeroporto	*airport*
Interior de um shopping center	*shopping_mall*
Estação de metrô	*metro_station*
Rua com pedestres	*street_pedestrian*
Praça pública	*public_square*
Rua com nível médio de tráfego	*street_traffic*
Transporte via bonde	*tram*
Transporte via ônibus	*bus*
Transporte via metrô subterrâneo	*metro*
Parque urbano	*park*

Fonte: do Autor.

3.1.1 Sistema de Classificação *Baseline*

Além do *dataset* fornecido pelo DCASE 2018 *Challenge*, também é disponibilizado um sistema de classificação *baseline*, para fins comparação com os sistemas dos participantes [37].

O sistema *baseline* é implementado utilizando redes neurais convolucionais e se baseia na arquitetura de um dos sistemas submetidos ao DCASE 2016 [38].

Como pré-processamento, os trechos de 10 segundos de cenas acústicas são convertidos em espectrogramas mel de formato (40, 500). Cada exemplo é convertido de estéreo para mono e, em seguida, são extraídas 40 bandas mel em um *log mel spectrogram*, com *frames* de 40 ms e *overlap* de 50%. Ao final desse processo, é feita a normalização dos atributos sobre todos os dados do conjunto de treinamento.

Na Tabela 3.2 é descrita a estrutura da rede neural *baseline*. As duas primeiras camadas são redes convolucionais, compostas uma sequência de camadas convolucional bidimensional, *batch normalization*, ReLU, *max pooling* 2D e *dropout*. As últimas camadas são do tipo MLP,

a primeira com ativação ReLU e *dropout*, e a camada de saída com ativação *softmax*. A operação de *flatten* simplesmente "achata" um tensor multidimensional, convertendo-o em um vetor unidimensional para uso na entrada da MLP.

Tabela 3.2: Estrutura da rede neural do sistema *baseline*.

Camada	**Detalhes**	
CNN #1	Conv2D	32 filtros, kernel shape: (7,7)
	Batch Normalization	-
	Ativação ReLU	-
	Max Pooling 2D	pool size: (5,5)
	Dropout	taxa: 30%
CNN #2	Conv2D	64 filtros, kernel shape: (7,7)
	Batch Normalization	-
	Ativação ReLU	-
	Max Pooling 2D	pool size: (4,100)
	Dropout	taxa: 30%
Flatten	-	-
MLP #1	MLP	unidades: 100, ativação: ReLU
	Dropout	taxa: 30%
Saída	MLP	unidades: 10, ativação: Softmax

Fonte: do Autor.

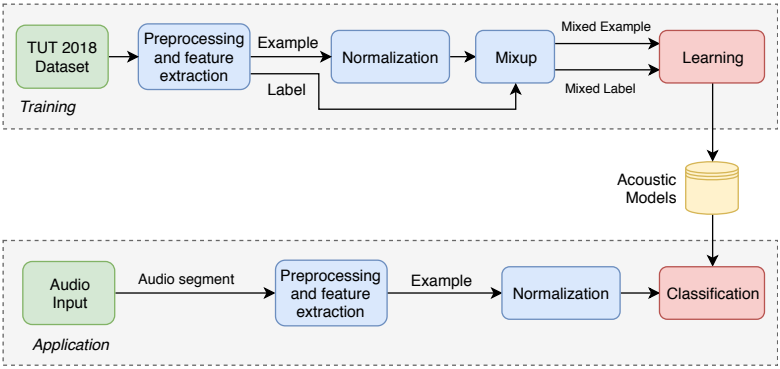
O sistema *baseline* é treinado durante 200 épocas com *minibatches* de tamanho 16. O otimizador usado é o Adam, com taxa de aprendizagem igual a 10^{-3}. Foi utilizado um conjunto de validação que consiste de aproximadamente 30% do conjunto de treinamento, selecionado de forma que ambos conjuntos não tivessem gravações do mesmo local. O desempenho do modelo é avaliado a cada época sobre o conjunto de validação, e o modelo de melhor desempenho é escolhido, alcançando um resultado de 59,7% ($\pm 0,7$) de acurácia no conjunto de teste [37].

Um código exemplo com a implementação é disponibilizado pelos autores em um repositório no *website* GITHUB [39]. Esse código fonte foi usado como referência nas etapas iniciais do desenvolvimento deste trabalho.

3.2 Sistema Proposto

Neste trabalho são propostas modificações ao sistema *baseline* a fim de obter um sistema com melhor desempenho. Na Figura 3.1, é apresentado o diagrama de blocos do sistema proposto, o qual é constituído das etapas de treinamento e aplicação. Na etapa de treinamento, é feito o pré-processamento e extração de atributos dos exemplos do TUT2018. Em seguida, os atributos são normalizados e é aplicado um *data augmentation* baseado na técnica *mixup* para realizar a aprendizagem de uma rede neural profunda. Na etapa de aplicação, o sistema realiza a classificação de sinais a partir dos modelos acústicos obtidos na etapa de treinamento.

Figura 3.1: Diagrama de blocos do sistema de classificação de cenas acústicas proposto.



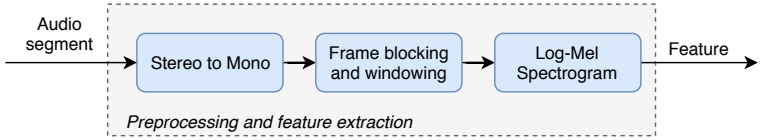
Fonte: do Autor.

3.2.1 Pré-processamento e Extração de Atributos

Similarmente ao sistema *baseline*, é feito o pré-processamento e a extração de atributos do conjunto de dados. Na Figura 3.2, é mostrado o diagrama de blocos do processo de pré-processamento e extração de atributos. Em tal processo, cada trecho de 10 segundos de áudio é convertido de estéreo para mono e, em seguida, é calculado o *log mel spectrogram* do sinal, com 100 bandas mel, *frames* de 40 ms e *overlap*

de 50%. A escolha do número de bandas mel a serem extraídas foi feita experimentalmente e é descrita na Seção 4.1.2.

Figura 3.2: Diagrama de blocos do pré-processamento e extração de atributos.



Fonte: do Autor.

3.2.2 Estrutura da Rede Neural

A estrutura de rede neural utilizada no sistema proposto é baseada em redes convolucionais, e está descrita na Tabela 3.3. As três primeiras camadas são convolucionais e, assim como no caso da rede do sistema *baseline*, são compostas por uma sequência de camadas convolucional bidimensional, *batch normalization*, ReLU, *max pooling* 2D e *dropout*. As últimas camadas são do tipo MLP, a primeira com ativação ReLU e *dropout*, e a camada de saída com ativação *softmax*.

Dentre as modificações em relação à estrutura *baseline*, é importante ressaltar a inclusão de uma camada convolucional adicional (CNN #3). Além disso, o formato do *kernel* dos filtros e o *pool size* foram alterados de forma a "ignorar" a primeira dimensão dos dados. Dessa maneira, as camadas convolucionais efetivamente só operam sobre o eixo temporal.

3.2.3 Etapa de Treinamento

No processo de treinamento do sistema, são treinados 5 modelos profundos em um esquema de validação cruzada *k-fold*. O otimizador utilizado é o Adam, com taxa de aprendizagem de 10^{-3}. Cada modelo é treinado por no máximo 600 épocas, com *minibatches* de 100 exemplos e *early stopping* com paciência de 100 épocas sobre a acurácia no conjunto validação. Como forma de *data augmentation*, foi utilizado o *mixup* com $\alpha = 0,2$.

Tabela 3.3: Estrutura da rede neural do sistema proposto.

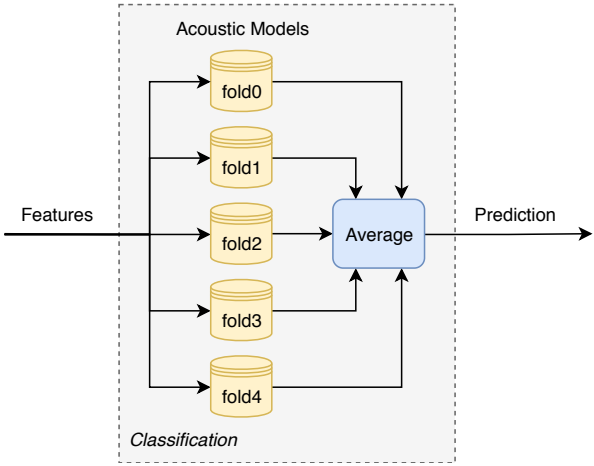
Camada	**Detalhes**	
CNN #1	Conv2D	32 filtros, kernel shape: (1,7)
	Batch Normalization	-
	Ativação ReLU	-
	Max Pooling 2D	pool size: (1,5)
	Dropout	taxa: 30%
CNN #2	Conv2D	64 filtros, kernel shape: (1,7)
	Batch Normalization	-
	Ativação ReLU	-
	Max Pooling 2D	pool size: (1,5)
	Dropout	taxa: 30%
CNN #3	Conv2D	32 filtros, kernel shape: (1,7)
	Batch Normalization	-
	Ativação ReLU	-
	Max Pooling 2D	pool size: (1,16)
	Dropout	taxa: 30%
Flatten	-	-
MLP #1	MLP	unidades: 100, ativação: ReLU
	Dropout	taxa: 30%
Saída	MLP	unidades: 10, ativação: Softmax

Fonte: do Autor.

3.2.4 Etapa de Aplicação

Na etapa de aplicação, o sistema realiza a classificação de sinais de áudio e pode ser avaliado sobre um conjunto de dados de teste. A Figura 3.3 ilustra o processo de classificação do sistema durante a etapa de aplicação. Com os 5 modelos treinados, o desempenho do sistema é avaliado a partir das predições de cada modelo sobre o conjunto de teste. É feita a média aritmética entre as predições dos modelos e calculada a acurácia sobre o conjunto de teste.

Figura 3.3: Diagrama de blocos da classificação durante a etapa de aplicação.



Fonte: do Autor.

3.2.5 Implementação

A implementação do sistema proposto foi realizada na linguagem de programação Python. Mais especificamente, o processo de extração de atributos das amostras de áudio foi realizado com o auxílio da biblioteca LibROSA [40], que fornece implementações de funções comumente usadas em análise de música e áudio, incluindo o cálculo de *log mel spectrogram*.

Além disso, para a definição da arquitetura e treinamento de redes neurais, utilizou-se as bibliotecas Tensorflow [41] e Keras [42]. A primeira permite que sejam definidas e executadas expressões matemáticas de algoritmos de aprendizagem de máquina, além de oferecer custo computacional reduzido, ao permitir o uso de GPUs para os cálculos. Keras fornece um nível mais alto de abstração sobre o Tensorflow, permitindo que modelos profundos sejam facilmente definidos e treinados.

CAPÍTULO 4

Resultados e Discussão

Neste capítulo são apresentados os resultados de experimentos realizados, assim como a definição do modelo final e um comparativo com o *baseline*.

4.1 Experimentos Realizados

Para avaliar o desempenho e algumas características do sistema de classificação desenvolvido foram realizados três experimentos. Em tais experimentos, foram avaliados os efeitos de técnicas de *batch normalization* e *dropout*, do número de bandas mel extraídas por *frame* e do uso de *data augmentation*.

Cada experimento avalia variações do sistema proposto, totalizando 10 análises. Como o sistema proposto utiliza um esquema de validação cruzada, deve-se treinar cinco modelos para cada avaliação realizada nos experimentos. Desse modo, considerando que uma das variações do sistema é a mesma entre os três experimentos, são treinados 40 modelos.

Além disso, não foram feitas avaliações de desempenho sobre o conjunto de treinamento, uma vez que o uso de *mixup* impede que os valores

de acurácia e a função custo sejam comparados entre os conjuntos de treinamento e validação na etapa de treinamento.

Os experimentos descritos anteriormente foram realizados na versão de avaliação do serviço de computação em nuvem *Google Cloud Platform*, que oferece uma máquina virtual *Debian Linux* com 2 CPUs virtuais, 13GB de RAM, 150GB de espaço de armazenamento em SSD e uma GPU *Nvidia Tesla K80*.

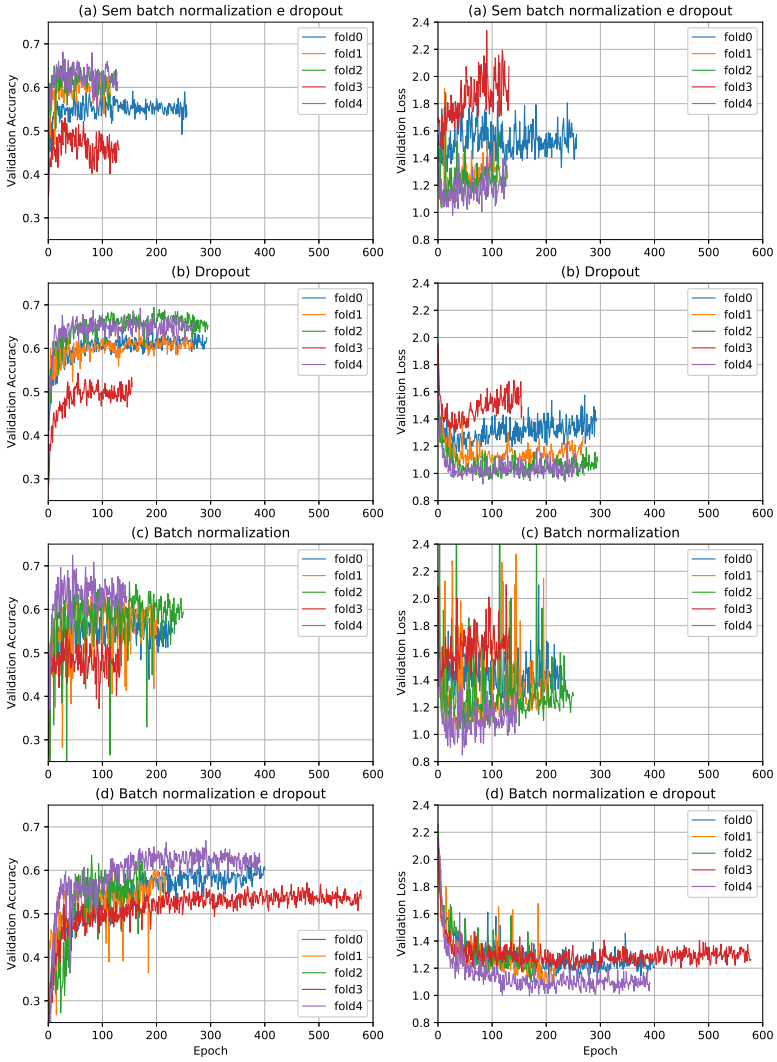
4.1.1 *Batch Normalization* e *Dropout*

No primeiro experimento, são avaliadas variações do sistema proposto quanto ao efeito do uso dos procedimentos de *batch normalization* e *dropout*. Assim, foram treinadas quatro arquiteturas que alternam a presença de *batch normalization* e *dropout* nas camadas da rede neural proposta. Todos os outros detalhes do sistema foram mantidos, e a arquitetura que utiliza ambos os procedimentos de *batch normalization* e *dropout* corresponde ao sistema proposto, sem modificações.

Na Figura 4.1, são mostrados a acurácia e o valor da função custo no conjunto de validação para cada época e *fold* de validação cruzada das arquiteturas avaliadas. Na arquitetura (a), ocorre *overfitting* de forma acentuada, enquanto que na arquitetura (c) há grandes variações de desempenho no decorrer das epócas de treinamento. O *overfitting* é reduzido e a otimização é estabilizada ao introduzir o *dropout*, na arquitetura (b), e mais ainda em conjunto com *batch normalization*, na arquitetura (d).

Na Tabela 4.1, é descrito o desempenho das arquiteturas treinadas para cada *fold* da validação cruzada, assim como o seu número de épocas e tempo de treinamento. A arquitetura (d), em média, teve uma otimização mais lenta que as outras arquiteturas, com um máximo de 579 épocas e mais de 3 horas de treinamento no *fold3*. A acurácia dos modelos ficou, no geral, acima de 60% para todos os conjuntos de validação, exceto o *fold3*, e foi observado um máximo de 72,45% no *fold4* da arquitetura (c).

Figura 4.1: Acurácia e função custo no conjunto de validação para cada época do treinamento dos modelos com variações no uso de *batch normalization* e *dropout*.



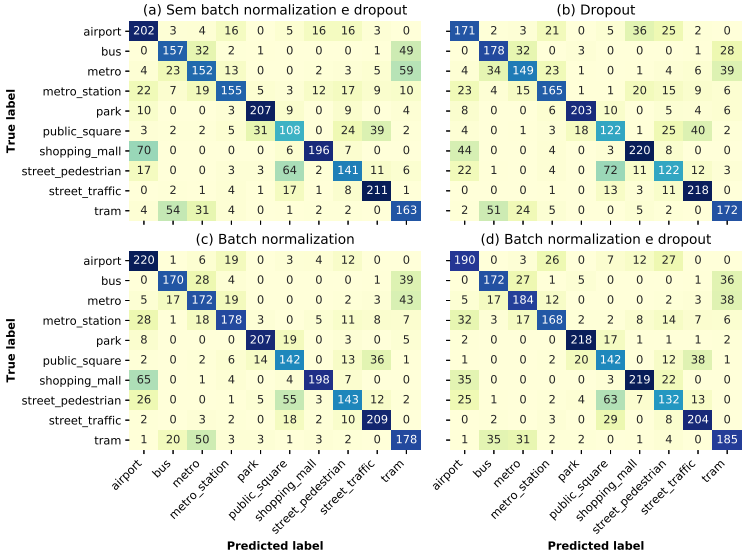
Fonte: do Autor.

Tabela 4.1: Detalhes dos modelos treinados com variações no uso de *batch normalization* e *dropout*. A acurácia e a função custo são relativos ao conjunto de validação.

(a) Sem *batch normalization* e *dropout*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	257	1h3m56s	59,14	1,38
1	116	29m3s	64,29	1,08
2	129	32m19s	66,15	1,07
3	132	33m5s	53,01	1,62
4	128	32m3s	68,13	0,98
(b) Com *dropout*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	294	1h19m	63,62	1,26
1	269	1h12m8s	64,70	1,07
2	296	1h20m6s	69,43	0,99
3	156	42m27s	54,32	1,34
4	271	1h13m37s	69,27	0,95
(c) Com *batch normalization*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	231	1h19m53s	60,28	1,25
1	205	1h10m55s	63,64	1,18
2	251	1h27m3s	66,31	1,06
3	134	46m44s	55,21	1,41
4	146	51m3s	72,45	0,85
(d) Com *batch normalization* e *dropout*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	401	2h21m19s	62,64	1,17
1	219	1h17m14s	60,44	1,18
2	181	1h4m12s	63,52	1,20
3	579	3h23m33s	57,17	1,21
4	392	2h19m4s	66,83	1.01

Fonte: do Autor.

Figura 4.2: Matriz de confusão das predições no conjunto de teste de sistemas com variações no uso de *batch normalization* e *dropout*.



Fonte: do Autor.

Na Figura 4.2, são mostradas as matrizes de confusão das predições sobre o conjunto de teste de cada variação do sistema proposto. Essa matriz é construída de forma que cada linha representa a classe predita e cada coluna a classe verdadeira. Os elementos da matriz indicam o número de exemplos classificados com um certo rótulo e o verdadeiro rótulo correspondente.

Na Tabela 4.2, são descritos os resultados de acurácia média e por classe das arquiteturas avaliadas neste experimento, no conjunto de teste. As arquiteturas (c) e (d) obtiveram as melhores acurácias médias, com 72,16% e 72,04%, respectivamente. Apesar de obter um desempenho ligeiramente inferior, a arquitetura (d) foi preferida por apresentar um comportamento mais estável que (c) durante o treinamento, como visto na Figura 4.1.

Tabela 4.2: Resultados de acurácia média e por cena acústica, no conjunto de teste, dos sistemas com variações no uso de *batch normalization* e *dropout*.

	Acurácia no conjunto de teste (%)			
Cena acústica	**(a) Sem batch normalization e dropout**	**(b) Com dropout**	**(c) Com batch normalization**	**(d) Com batch normalization e dropout**
airport	76,23	64,53	83,02	71,70
bus	64,88	73,55	70,25	71,07
metro	58,24	57,09	65,90	70,50
metro_station	59,85	63,71	68,73	64,86
park	85,54	83,88	85,54	90,08
public_square	50,00	56,48	65,74	65,74
shopping_mall	70,25	78,85	70,97	78,49
street_pedestrian	57,09	49,39	57,89	53,44
street_traffic	85,77	88,62	84,96	82,93
tram	62,45	65,90	68,20	70,88
Média	67,20	68,31	72,16	72,04

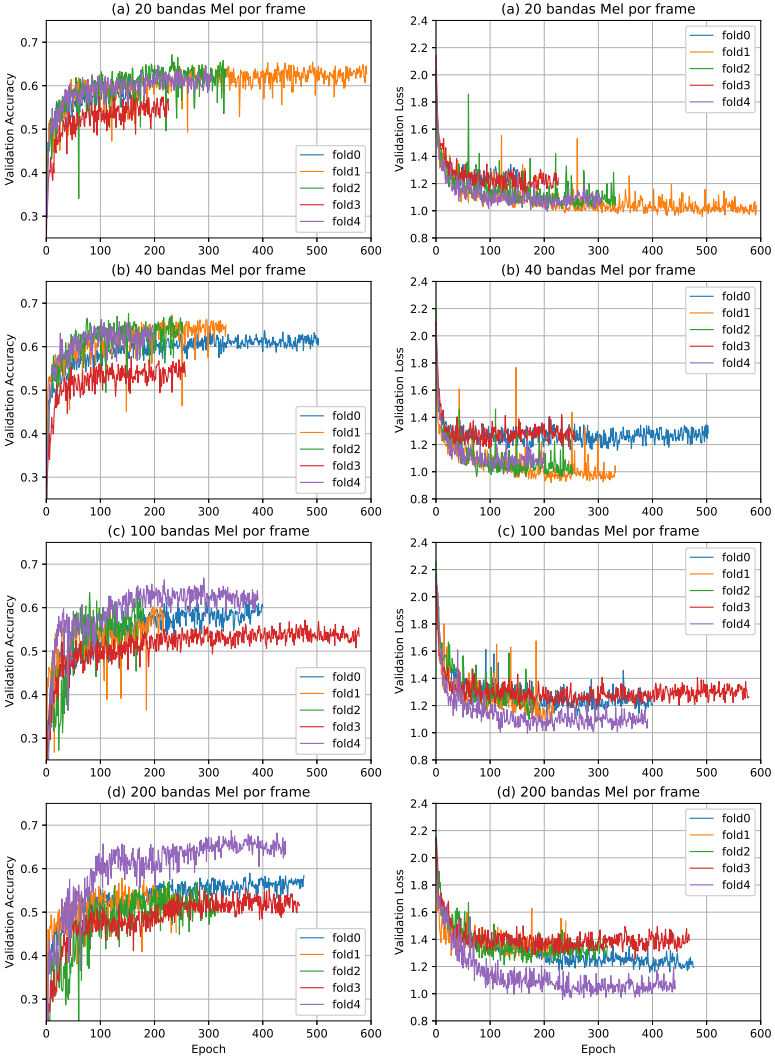
Fonte: do Autor.

4.1.2 Número de Bandas Mel

No seguinte experimento, são avaliadas variações do sistema proposto quanto ao número de bandas mel extraídas, por *frame* do espectrograma. Assim, foram treinadas quatro arquiteturas com 20, 40, 100 e 200 bandas mel. O número de parâmetros varia entre as arquiteturas, visto que a quantidade de pesos é proporcional ao número de bandas mel, pois a rede neural proposta contém camadas MLP. Outros detalhes do sistema foram mantidos, incluindo o tamanho dos *frames* e *overlap* usados na extração de atributos. A arquitetura que utiliza 100 bandas mel corresponde ao sistema proposto, sem modificações.

Na Figura 4.3, são mostrados a acurácia e o valor da função custo no conjunto de validação para cada época e *fold* de validação cruzada. Em geral, o comportamento do processo de treinamento foi similar entre os modelos, com *overfitting* ocorrendo da mesma forma. Entretanto, certos modelos são treinados mais facilmente para *folds* específicos, por exemplo, as arquiteturas com 20 e 40 bandas mel se saem melhor com o *fold1*, enquanto que as arquiteturas com 100 e 200 bandas mel se saem melhor com o *fold4*.

Figura 4.3: Acurácia e função custo no conjunto de validação para cada época do treinamento dos modelos com variações no número de bandas mel.



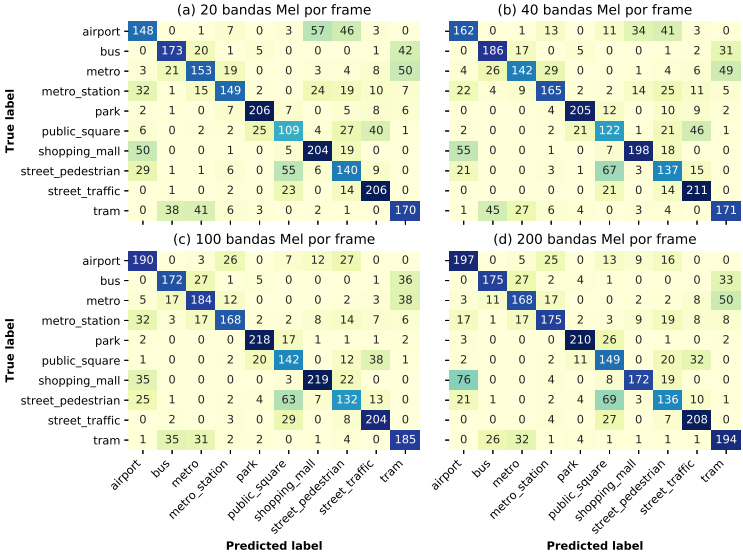
Fonte: do Autor.

Tabela 4.3: Detalhes dos modelos treinados com variações no número de bandas mel. A acurácia e a função custo são relativos ao conjunto de validação.

(a) 20 bandas mel por *frame*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	160	14m17s	61,09	1,14
1	593	52m54s	65,44	0,96
2	333	29m57s	67,13	1,01
3	227	20m34s	57,90	1,16
4	307	27m48s	64,71	0,99
(b) 40 bandas mel por *frame*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	504	1h17m4s	63,70	1,25
1	333	51m18s	67,32	0,94
2	253	38m59s	67,62	0,97
3	258	39m47s	57,74	1,22
4	202	31m23s	65,93	1,03
(c) 100 bandas mel por *frame*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	401	2h21m19s	62,64	1,17
1	219	1h17m14s	60,44	1,18
2	181	1h4m12s	63,52	1,20
3	579	3h23m33s	57,17	1,21
4	392	2h19m4s	66,83	1,01
(d) 200 bandas mel por *frame*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	477	5h42m16s	58,97	1,18
1	241	2h53m16s	57,82	1,29
2	318	3h49m24s	57,05	1,22
3	469	5h38m59s	55,70	1,25
4	443	5h20m3s	68,79	0,97

Fonte: do Autor.

Figura 4.4: Matriz de confusão das predições no conjunto de teste de sistemas com variações no número de bandas mel por *frame*.



Fonte: do Autor.

Na Tabela 4.3, é descrito o desempenho dos modelos para cada *fold* da validação cruzada, assim como o número de épocas e o tempo de treinamento. Apesar de um número de épocas similar, as arquiteturas com maior número de bandas mel, e consequentemente mais parâmetros, têm um tempo de treinamento mais elevado. Outro ponto relevante é o baixo desempenho ao utilizar 200 bandas mel, o que sugere limitações da arquitetura proposta em relação ao número de dimensões dos atributos utilizados, e confirma que redes com muitos parâmetros são mais difíceis de treinar.

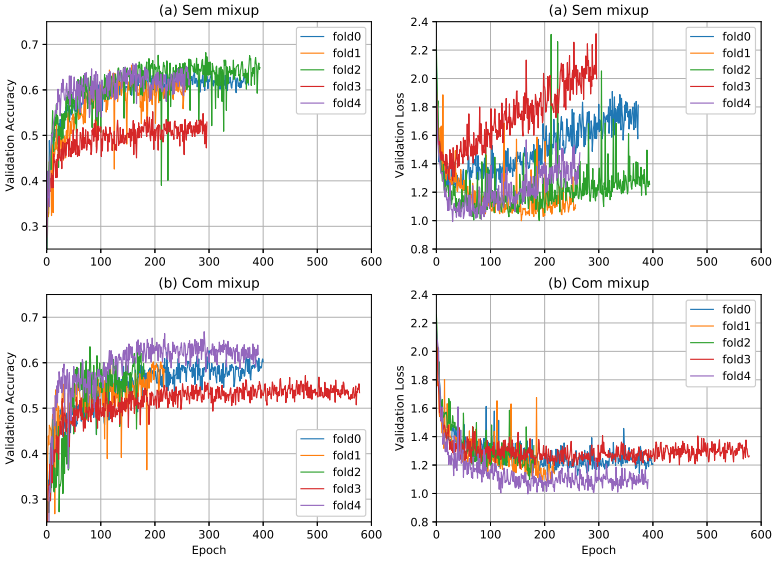
Na Figura 4.4, são mostradas as matrizes de confusão das predições de cada sistema sobre o conjunto de teste e, na Tabela 4.4, são descritos os resultados de acurácia média e por classe das arquiteturas avaliadas neste experimento, no conjunto de teste. É possível notar que o aumento do número de bandas mel, até certo ponto, beneficia o desempenho do sistema. Além disso, os resultados do sistema que utiliza 40 bandas mel sugerem que as modificações propostas ao sistema *baseline* são efetivas.

Tabela 4.4: Resultados de acurácia média e por cena acústica, no conjunto de teste, dos sistemas com variações no número de bandas mel por *frame*.

	Acurácia no conjunto de teste (%)			
Cena acústica	**20 mel *bands***	**40 mel *bands***	**100 mel *bands***	**200 mel *bands***
airport	55,85	61,13	71,70	74,34
bus	71,49	76,86	71,07	72,31
metro	58,62	54,41	70,50	64,37
metro_station	57,53	63,71	64,86	67,57
park	85,12	84,71	90,08	86,78
public_square	50,46	56,48	65,74	68,98
shopping_mall	73,12	70,97	78,49	61,65
street_pedestrian	56,68	55,47	53,44	55,06
street_traffic	83,74	85,77	82,93	84,55
tram	65,13	65,52	70,88	74,33
Média	65,85	67,47	72,04	70,85

Fonte: do Autor.

Figura 4.5: Acurácia e função custo no conjunto de validação para cada época do treinamento dos modelos com variações no uso de *data augmentation*.



Fonte: do Autor.

4.1.3 *Data Augmentation*

Neste experimento, é avaliado o efeito do uso de *data augmentation* do tipo *mixup* no sistema proposto. Assim, são treinadas duas variações do sistema: com e sem *mixup*. Todos os detalhes da rede neural proposta foram mantidos, apenas sendo aplicado, ou não, *data augmentation* nos *minibatches* durante a etapa de treinamento. A arquitetura que utiliza *mixup* corresponde ao sistema proposto, sem modificações.

Na Figura 4.5, são mostrados a acurácia e o valor da função custo no conjunto de validação para cada época e *fold* de validação cruzada das arquiteturas avaliadas. É possível notar um *overfitting* acentuado no treinamento sem *mixup*, e que o uso de *mixup* reduz significativamente o *overfitting* dos modelos. Esse efeito é mais claro nos gráficos do valor da função custo.

Na Tabela 4.5, é descrito o desempenho dos modelos para cada *fold* da validação cruzada, assim como o número de épocas e o tempo de treinamento. Apesar da redução de *overfitting* com o uso de *mixup*, o treinamento sem *mixup* obteve resultados melhores no conjunto de validação em 3 dos 5 *folds*. Entretanto, isso pode ter sido causado por um viés do modelo em relação aos dados do conjunto de validação, visto que é usado *early stopping* com monitoramento da acurácia de validação para determinar a parada do algoritmo.

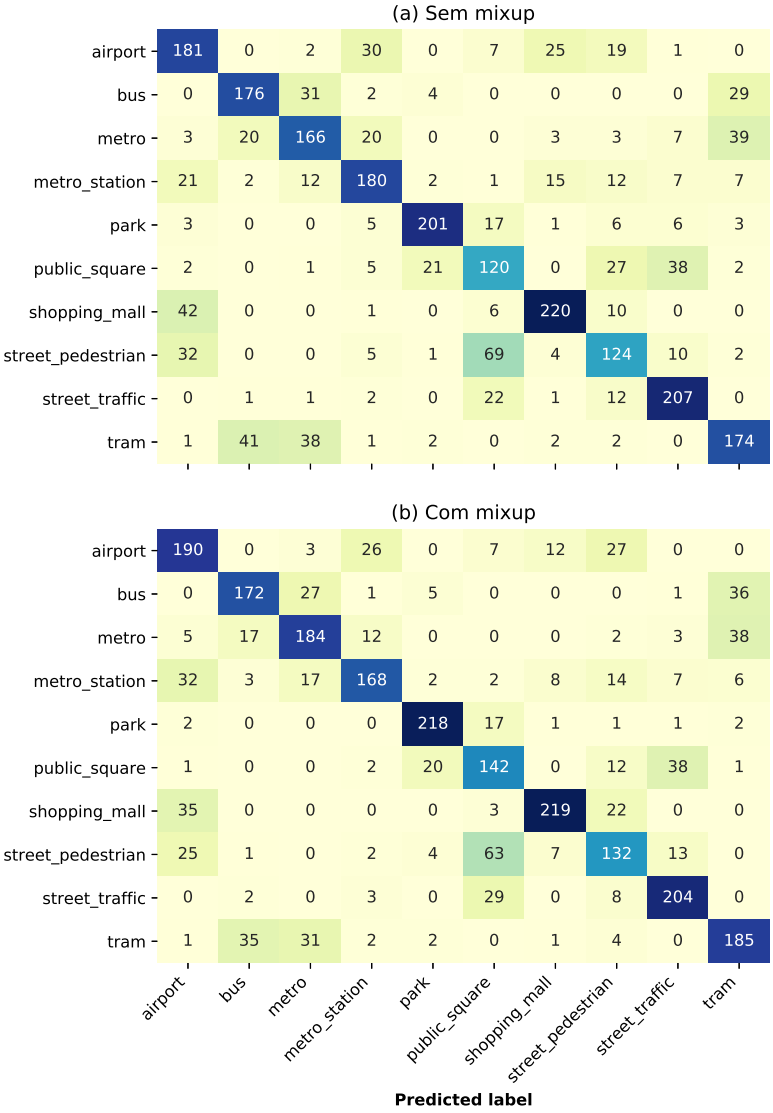
Na Figura 4.6, são mostradas as matrizes de confusão das predições de cada sistema sobre o conjunto de teste e, na Tabela 4.6, são descritos os resultados de acurácia média e por classe das arquiteturas avaliadas neste experimento, no conjunto de teste. É observada uma melhoria de aproximadamente 2,5% na acurácia no sistema proposto ao utilizar a técnica de *mixup data augmentation*.

Tabela 4.5: Detalhes dos modelos treinados com variações no uso de *data augmentation*. Acurácia e função custo relativos ao conjunto de validação.

(a) Sem *mixup*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	374	2h11m42s	64,36	1,52
1	258	1h31m2s	65,44	1,00
2	395	2h19m39s	68,20	1,14
3	297	1h44m57s	55,21	1,69
4	266	1h34m17s	65,77	1,17
(b) Com *mixup*				
Fold	**Épocas**	**Tempo de treinamento**	**Acurácia (%)**	**Função custo**
0	401	2h21m19s	62,64	1,17
1	219	1h17m14s	60,44	1,18
2	181	1h4m12s	63,52	1,20
3	579	3h23m33s	57,17	1,21
4	392	2h19m4s	66,83	1,01

Fonte: do Autor.

Figura 4.6: Matriz de confusão das predições no conjunto de teste de sistemas com variações no uso de *data augmentation*.



Fonte: do Autor.

Tabela 4.6: Resultados de acurácia média e por cena acústica, no conjunto de teste, dos sistemas com variações no uso de *data augmentation*

	Acurácia (%)	
Cena acústica	**Sem Mixup**	**Com Mixup**
airport	68,30	71,70
bus	72,73	71,07
metro	63,60	70,50
metro_station	69,50	64,86
park	83,06	90,08
public_square	55,56	65,74
shopping_mall	78,85	78,49
street_pedestrian	50,20	53,44
street_traffic	84,15	82,93
tram	66,67	70,88
Média	69,46	72,04

Fonte: do Autor.

4.2 Comparativo entre *Baseline* e o Sistema Proposto

Comparado ao sistema *baseline*, o sistema de classificação proposto apresenta melhorias significativas no desempenho. Na Tabela 4.7, são descritos os resultados de acurácia média e por cena acústica, no conjunto de teste, de ambos sistemas. O sistema proposto supera a acurácia do *baseline* em quase todas a cenas acústicas avaliadas, tendo pior desempenho somente na cena *airport*. O aumento de mais de 10% na acurácia sobre o *baseline* comprova a eficácia do sistema proposto.

Tabela 4.7: Resultados de acurácia média e por cena acústica, no conjunto de teste, dos sistemas *baseline* e proposto.

	Acurácia (%)	
Cena acústica	**Baseline**	**Proposto**
airport	72,9	71,7
bus	62,9	71,1
metro	51,2	70,5
metro_station	55,4	64,9
park	79,1	90,1
public_square	40,4	65,7
shopping_mall	49,6	78,5
street_pedestrian	50,0	53,4
street_traffic	80,5	82,9
tram	55,1	70,9
Média	59,7	72,0

Fonte: do Autor.

CAPÍTULO 5

Conclusão

Neste trabalho, foi apresentado um sistema de classificação de cenas acústicas baseado em técnicas de aprendizagem profunda. Foram analisados algoritmos relevantes no contexto de tarefas de classificação, assim como técnicas de extração de atributos aplicadas à cenas acústicas. No desenvolvimento do sistema foi escolhido um conjunto de dados adequado para a realização do treinamento de algoritmos de aprendizagem profunda. Foram propostas alterações em um sistema de classificação de cenas acústicas *baseline*, incluindo o uso de *data augmentation* e um maior número de atributos, além de modificações na estrutura da rede neural utilizada. Também foram analisados os efeitos de variações nas características do sistema. Por fim, obteve-se melhorias significativas no desempenho do sistema proposto, aumentando a acurácia do sistema de 59,7%, no *baseline*, para 72,04%, no sistema proposto.

Trabalhos futuros

A fim de melhorar o desempenho do sistema proposto, outras técnicas de aprendizagem profunda podem ser analisadas, como redes neurais recorrentes (RNNs) e redes adversárias generativas (GANs). Além disso,

o sistema desenvolvido neste trabalho pode ser utilizado como base para uma futura submissão ao DCASE 2019 *Challenge*. Por fim, as ideias expostas neste trabalho também podem auxiliar na implementação de sistemas acústicos de monitoramento urbano.

Referências bibliográficas

[1] Tuomas Virtanen, Mark D Plumbley, and Dan Ellis. *Computational analysis of sound scenes and events*. Springer, 2018.

[2] S. Chachada and C. . J. Kuo. Environmental sound recognition: A survey. In *2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pages 1–9, Oct 2013.

[3] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu. Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions. *CoRR*, abs/1712.05884, 2017.

[4] Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Katya Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769, 2017.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural

networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[6] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[10] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[12] MATLAB. Convolution and polynomial multiplication (conv). `https://www.mathworks.com/help/matlab/ref/conv.html`, acesso em Dezembro de 2018.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, 2009.

[15] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[17] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[19] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[22] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[23] Dimitris G Manolakis and Vinay K Ingle. *Applied digital signal processing: theory and practice*. Cambridge University Press, 2011.

[24] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[25] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The HTK book, version 3.4. *Cambridge University*, Mar 2009.

[26] J. Salamon and J. P. Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283, March 2017.

[27] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[28] Karol J. Piczak. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pages 1015–1018. ACM Press, 10 2015.

[29] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[30] Shayan Gharib, Honain Derrar, Daisuke Niizumi, Tuukka Senttula, Janne Tommola, Toni Heittola, Tuomas Virtanen, and Heikki Huttunen. Acoustic scene classification: A competition review. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2018.

[31] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M.D. Plumbley. Detection and classification of acoustic scenes and events. *Multimedia, IEEE Transactions on*, 17(10):1733–1746, October 2015.

[32] A. Mesaros, T. Heittola, E. Benetos, P. Foster, M. Lagrange, T. Virtanen, and M. D. Plumbley. Detection and classification of acoustic scenes and events: Outcome of the dcase 2016 challenge. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(2):379–393, February 2018.

[33] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen. DCASE 2017 challenge setup: Tasks, datasets and baseline system. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, pages 85–92, November 2017.

[34] Seongkyu Mun, Sangwook Park, David Han, and Hanseok Ko. Generative adversarial network based acoustic scene training set augmentation and selection using SVM hyper-plane. Technical report, DCASE2017 Challenge, September 2017.

[35] Yoonchang Han and Jeongsoo Park. Convolutional neural networks with binaural representations and background subtraction for acoustic scene classification. Technical report, DCASE2017 Challenge, September 2017.

[36] Toni Heittola, Annamaria Mesaros, and Tuomas Virtanen. TUT Urban Acoustic Scenes 2018, Development dataset. `https://doi.org/10.5281/zenodo.1228142`, April 2018.

[37] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. A multi-device dataset for urban acoustic scene classification. `https://arxiv.org/abs/1807.09840`, 2018.

[38] Michele Valenti, Stefano Squartini, Aleksandr Diment, Giambattista Parascandolo, and Tuomas Virtanen. A convolutional neural network approach for acoustic scene classification. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1547–1554. IEEE, 2017.

[39] Toni Heittola. DCASE 2018 Baseline system. `https://github.com/DCASE-REPO/dcase2018_baseline`, 2018.

[40] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 18 – 25, 2015.

[41] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay

Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[42] François Chollet et al. Keras. `https://keras.io`, 2015.